

# User's Manual for **Pelegant**

Program Version 17.1.1  
Advanced Photon Source

Yusong Wang, Michael Borland, Robert Soliday

April 3, 2007

## 1 Highlights of What's New in Version 17.1.1

Most of efforts were put on the parallelization of elements with collective effects, which include wakefields and coherent synchrotron radiation. For a lattice dominated by collective effects, the program scales well on 30 processors. Performance improvement for a larger number of processor is on-going.

### 1.1 The new parallelized elements

CHARGE CLEAN CSRCSBEND CSRDRIFT FRFMODE FTRFMODE ILMATRIX RFMODE  
TRFMODE TRWAKE TWISSELEMENT WAKE ZLONGIT ZTRANSVERSE

### 1.2 Tracking rf cavity with wakefields is available

This function was disabled in the previous version because of the synchronization problem when calling a serial element from a parallelized element. It becomes possible after the wake elements are parallelized.

## 2 Introduction

**Pelegant** stands for “parallel elegant,” which is a parallelized version of **elegant** [1]. Written in the C programming language with MPICH, the **Pelegant** has been successfully ported to several clusters and supercomputers, such as the “weed” cluster (a heterogeneous system of 100 CPUs) at Advanced Photon Source (APS), and the Jazz cluster (350 Intel Xeon CPUs) at Argonne National Lab (ANL) and the BlueGene/L supercomputer (1024 dual PowerPC 440 nodes) at Argonne National Lab. Thanks to careful design in parallelization and good architecture of the serial **elegant**, the **Pelegant** achieves very good performance. For example, for a simulation of  $10^5$  particles in APS including symplectic element-by-element tracking, accelerating cavities, and crab cavities, the simulation time was reduced from 14.3 days to 42 minutes on 512 CPUs of the BlueGene/L (BG/L) supercomputer. The speedup for this particular simulation is 484 with efficiency near 95%.

This document describes how to build **Pelegant**, run the code and optimize the performance. Finally, appendices are included describing which elements have been parallelized and which commands have been used in the regression tests. The user should be familiar with the User's Manual for **elegant** before reading this document.

## 3 Building **Pelegant**

The steps in building **Pelegant** on Linux are as follows:

1. Install MPICH (1 or 2) [2]. Your cluster administrator will need to do this.
2. If you are using an x86 processor, set the environment variables `HOST_ARCH` and `EPICS_HOST_ARCH` to `linux-x86`. If you are using a ppc processor, you can set these variables to `linux-ppc`.
3. If EPICS/Base is already installed on your computer, you can skip to the next step. Pelegant is built using the EPICS/Base configure files available from the OAG web site at APS. You will need to unpack this to create `epics/base/configure`. Go to the `epics/base` directory and type “make.”
4. Next you will need to download the EPICS extensions configure files from the OAG web site. This will unpack to create `epics/extensions/configure`. Go to the `epics/extensions/configure` directory and type “make.”
5. Download the latest SDDS source code from the OAG web site. This will unpack to create `epics/extensions/src/SDDS`. Go to this directory and type “make.”
6. Download the OAG configure files from the OAG web site. This will unpack to create `oag/apps/configure`. Go to this directory and type “make.”
7. Download the **Pelegant** source code from the OAG web site. This will unpack to create `oag/apps/src/elegant`.
8. Set the path for your installation of MPICH in `Makefile.OAG` in `oag/apps/src/elegant`.
9. From `oag/apps/src/elegant`, run the following command to build **Pelegant**:

```
make Pelegant
```

**Pelegant** should now exist at `oag/apps/bin/linux-x86/Pelegant`. (To build the serial version, just type “make.” This will also build related software.)

**elegant** and **Pelegant** share the same source code. Makefile will decide which part of code will be compiled according to the binary file (either **Pelegant** or **elegant**) you want to build.

If you have any question about installing the **Pelegant**, please send an email to [soliday@aps.anl.gov](mailto:soliday@aps.anl.gov).

## 4 Running a simulation with Pelegant

Running a parallel job is just as easy as starting a serial job, while the time spent on a job can be reduced from several days to hours, or even minutes. **Pelegant** has been tested under both MPICH-1 and MPICH-2. We suggest using MPICH2 if possible, as it shows better stability in our regression test. In the future, we may also add new features available in MPI-2, such as parallel I/O, remote memory operations, to improve the performance of **Pelegant** for some simulations.

### 4.1 Running Pelegant with MPI command

An examples directory is available under the **elegant** directory. Users can run a simulation with a given lattice and input files according to the version of the MPI implementation.

For example, we can run the parallel **elegant** on 11 processors (10 working processors) with the following commands: <sup>1</sup>

- 1) For MPI-1, we can use the following syntax:  

```
mpirun -np 11 Pelegant manyParticles_p.ele
```

---

<sup>1</sup>For the weed cluster in APS, all the MPI-related commands have been put in the options of `csub` command. User could just type:

```
csub -mpich2 11 Pelegant manyParticles_p.ele
```

- 2) For MPI-2, `mpiexec` is strongly encouraged to start MPI programs, e.g.,  
`mpiexec -np 11 Pelegant manyParticles.p.ele`

One can run another simulation with the serial version of **elegant** for comparison at the same time:

```
elegant manyParticles.s.ele
```

The contents of the input files are same, while “`_p`” and “`_s`” are corresponding to the input files of **Pelegant** and **elegant**, respectively. User should check the **elegant** manual to prepare the input file.

In principle, users can run simulations on any number of processors. We have tested the program with 1024 nodes on the BG/L supercomputer at Argonne National Lab. However, the number of processors can not be more than the number of particles to track, as it will not use the resource efficiently.

## 4.2 Validating the result

The simulations above were finished in around 1 minute for **Pelegant** and 10 minutes for **elegant** on the AMD Athlon nodes of the weed cluster at the APS. To convince ourselves the results of the two versions of **elegant** are same, one can compare the output files with the `sddsdiff` command, which should be available in the SDDS toolkit. For example, to examine the particle coordinates at interior points, we can type:

```
sddsdiff manyParticles.p.w2 manyParticles.s.w2
```

which should return that two results are identical.

Validating a parallel program against a uniprocessor program with the requirement of bitwise identical result is notoriously difficult [3], as we may meet some new problems raised from parallel computing, such as different ordering of summations, or a nonscalable random number generator. Although the simulation results with the discrepancies should conform to IEEE 754 within some tolerance, more consistent results can be expected with a more accurate numerical algorithm, such as Kahan’s summation formula [4], which has been employed in both serial and parallel versions of **elegant**.

We ran a regression test of 92 cases and validated the results of **Pelegant** with those of **elegant**. As the random number sequences generated by one CPU and multiple CPUs usually are not the same, some test examples can’t be validated by comparing the results of **elegant** and **Pelegant**. Those examples have been validated either by mathematical formulae or their physical meaning.

## 5 Using Pelegant efficiently

As the **elegant** parallelization is an on-going project, so far we have parallelized 74 out of 95 elements for tracking simulations. The elements that have been parallelized are listed at the end of this manual. If most time-intensive parts of a simulation have been parallelized, we can expect good speedup and efficiency. If a simulation is dominated by elements that have not been parallelized, e.g., wakefield elements, we recommend using the serial **elegant** at this point.

### 5.1 Parallelization overview

To help users run simulations with **Pelegant** more efficiently, we would like to introduce our parallelization approach briefly. We parallelize **elegant** using a master/slaves (manager/workers) model. The time-intensive tracking parts of **elegant** are being parallelized gradually. The other parts are done (redundantly) by all the processors, which is acceptable since those processors have already been allocated to a particular **Pelegant** run. We divide the beamline elements into four classes:

1. Parallel element: only the slave processors will do the tracking. Each slave is responsible for a portion of particles.

2. MP (multiprocessor) algorithm: the master will participate in the tracking, but it only gets the result of collective computations (e.g., sum, average) from the slaves, without doing any computations itself.
3. Uniprocessor element: must be done by master (for now) and modifies particle coordinates. An example would be wakefield elements.
4. Diagnostic: same as the uniprocessor element, but doesn't change particle coordinates.

A flag was added to **elegant**'s dictionary for each beamline element to identify its classification. The master is responsible for gathering and scattering particles as needed according to this classification. Communications are minimized to achieve the best efficiency of parallelization. For example, it is not necessary to communicate the coordinates of particles between master and slaves when tracking through two continuous parallel elements. Similarly, we only need to gather particle coordinates from slaves to master (without subsequent scattering) when the particles go through a diagnostic element, such as a WATCH point with coordinate output.

## 5.2 Achieving high performance

In our master/slave model, the master will be responsible for I/O operations and communicating with the slave processors only, i.e., it will not do the tracking for most of the elements. As a result, 10 or more processors are recommended when running simulation with **Pelegant**. To run simulations efficiently, we also suggest when possible that the user arrange all serial elements in a continuous sequence, which will minimize the communication overhead for gathering and scattering particles. This will be unnecessary in the future when all of the elements are parallelized.

By default, **Pelegant** is built in such a way that it does load balancing after each pass through the accelerator. This is particularly important when the user does not have exclusive use of the nodes. When running **Pelegant** in an environment where only one user is allowed to run a job on a computer node at a time, then **Pelegant** can be optimized by defining the compiler flag `CHECKFLAGS=1` in the `Makefile.OAG`. In this case, the load balance will be checked only after the first turn or when the particle number is changed, instead of every turn.

For ANL users who need to run simulations that would normally take several weeks or months with serial **elegant**, we can provide help to perform runs on the Jazz cluster (350 nodes, each with a 2.4 GHz Pentium Xeon) or the BlueGene/L supercomputer (1024 dual PowerPC 440 700MHz 512MB nodes) at ANL. **Pelegant** is pre-built and available on both systems.

## 6 What is not supported

In our regression test for **Pelegant**, we excluded two types of tests, which are not supported in this version of **Pelegant**:

1. All of the tests tracking beam with one particle (or the number of particles is less than the number of processors) were excluded, as such a simulation will not benefit from the parallelization approach we employed.
2. The third type of tests we excluded are those needing slice analysis or an aperture search. They are not supported in **Pelegant** at present.

As some elements have not been parallelized in this version of **Pelegant**, it is possible that **Pelegant** will hang and never stop for some particular simulations, which happens when calling a parallelized function from a serial element. We suggest running a simulation with a small workload first before trying the final time-intensive simulation. Also, use of WATCH elements with `FLUSH_INTERVAL=1` can be helpful in verifying that progress is being made. We listed the commands that have been tested in the appendices. Users can report bugs with all the input files to [ywang25@aps.anl.gov](mailto:ywang25@aps.anl.gov) or [borland@aps.anl.gov](mailto:borland@aps.anl.gov).

## 7 Appendices

### 7.1 Elements that have been parallelized in Pelegant

The particular physical elements that take advantage of parallel computation in the present version of the code are:

1. Drift spaces, dipoles, quadrupoles, sextupoles, higher multipoles, dipole correctors, and wiggler magnets, whether symplectically integrated or modeled with a transport matrix (up to 3rd order). Symplectically integrated elements can optionally include both quantum and classical synchrotron radiation effects.
2. Radio frequency cavities, including accelerating and deflecting cavities, with constant field amplitude.
3. Accelerating cavities with phase, voltage, and frequency modulation or ramping.
4. Beam collimating and scraping elements.
5. Field-map integration elements, such as dipoles, x-y dependent maps, and solenoids.
6. Reference energy matching points.
7. Beam watch points, which may involve parallel computation of beam moments or serial dumping of particle coordinates.
8. Scattering elements, including lumped-element simulation of synchrotron radiation.
9. Elements with collective effects, including wakefields and coherent synchrotron radiation.

Here is an explicit list of the elements:

ALPH BEND BEND BMAPXY CHARGE CLEAN CSBEND CSRCSBEND CSRDRIFT CWIGGLER DRIFT ECOL EDRIFT EMATRIX ENERGY FLOORELEMENT FMULT FRFMODE FTRFMODE HCOR HMON HVCOR ILMATRIX KICKER KPOLY KQUAD KSBEND KSEXT LMIRROR LTHINLENS MAGNIFY MALIGN MAPSOLENOID MARK MATR MATTER MAXAMP MODRF MONI MULT NIBEND NISEPT PEPPOT QFRING QUAD RAMPRF RCOL RECIRC REFLECT RFCA RFCW RFDF RFMODE RFTM110 ROTATE SAMPLE SCATTER SCMULT SCRAPER SEXT SOLE SREFFECTS STRAY TAYLORSERIES TRFMODE TRWAKE TUBEND TWISSELEMENT VCOR VMON WAKE WIGGLER ZLONGIT ZTRANSVERSE

### 7.2 Commands that have been tested for Pelegant

In addition to tracking, the following features of **elegant** may be used in the parallel version:

1. Optimization with tracking. In this case, the optimization is supervised at the serial level while the tracking is done in parallel.
2. Computation and output of Twiss parameters and transport matrices along a beamline.
3. Computation and output of beam statistics along a beamline.
4. Alteration of element properties, loading of element parameters from external files, and transmutation of element types.
5. Scanning of element properties in loops. In this case, the scanning is supervised at the serial level while the tracking is done in parallel.
6. Use of internally generated or externally supplied particle distributions.
7. Addition of random errors to accelerator components.

## 8. Computation and output of closed orbits.

Here is an explicit list of the commands:

```
alter_elements  
bunched_beam  
closed_orbit  
error_control  
error_element  
link_elements  
load_parameters  
matrix_output  
optimization_setup  
optimization_term  
optimization_variable  
optimize  
run_control  
run_setup  
sasefel  
save_lattice  
sdds_beam  
stop  
subprocess  
track  
transmute_elements  
twiss_output  
vary_element
```

## References

- [1] M. Borland, “elegant: A Flexible SDDS-Compliant Code for Accelerator Simulation,” Advanced Photon Source LS-287, September 2000.
- [2] MPICH Home Page. <http://www-unix.mcs.anl.gov/mpi/>
- [3] W. D. Gropp, “Accuracy and Reliability in Scientific Computing,” chapter Issues in Accurate and Reliable Use of Parallel Computing in Numerical Program. SIAM, 2005.
- [4] D. Goldberg, “What every computer scientist should know about floating-point arithmetic,” ACM Computing Surveys, 23(1):5–48, March 1991.